Modeling and Verification of Reconfigurable Distributed Control Systems

Friederike Bruns*, Bianca Wiesmayr†, Felix Schmid†, Lisa Wünderlich*, and Andreas Rauh*
* DCIS, Carl von Ossietzky Universität, Ammerländer Heerstraße 114-118, Oldenburg, Germany, email: first.last@uol.de
† LIT, Johannes Kepler University, Altenbergerstr. 69, Linz, 4040, Austria, email: first.last@jku.at

***Abstract*—Distributed control systems increasingly rely on heterogeneous, mixed-criticality communication that must meet diverse real-time requirements, adapt to dynamic modes, and ensure safety and correctness. Traditional design workflows often treat communication and control logic separately, limiting early verification and complicating system integration. The proposed design flow supports offline verification of reconfigurable communication architectures while ensuring predictable execution under strict timing constraints. Communication is modeled explicitly through logical messages and physical channels in a contract-based design flow, allowing integration of real-time or quality-of-service properties into the application model. This enables the specification of assumption-guarantee contracts, supports early syntactic and semantic checks, and incorporates dynamic validation of time traces. A battery production case study demonstrates communication heterogeneity and dynamic operating modes in a tool-supported modeling environment.**

***Index Terms*—Mixed-Criticality, Network Communication, Flexible Reconfiguration, Contract-Based Design**

I. Introduction

In robotic assembly lines, robots and sensors cooperate in a shared workspace. When safety-relevant events occur, such as human presence detection or component failures, the system must both react within hard real-time bounds and reconfigure its control structure by changing coordination modes or isolating affected components. Reconfiguration capabilities are essential to continuously update production processes [11], [15]. However, this reconfiguration must be completed within guaranteed time limits to avoid unsafe intermediate states. Existing solutions, such as replacing mechatronic components, focus on supporting software engineers [9] or offer online reconfiguration mechanisms for replacing the control software without disturbing the overall system [16]. Other solutions provide high-level descriptions of the expected control process to orchestrate modules of manufacturing systems [19] or focus on online reconfiguration after actuator faults [14].

In distributed systems, reconfiguration mechanisms have to cover network communication, when re-distributing control software to different devices during reconfiguration. Particularly challenging for industrial distributed control systems is the coexistence of multiple network requirements such as security, throughput, and real-time capabilities [20]. While plug-and-produce approaches enable dynamic reconfiguration of the communication infrastructure [5], [6], they often conflict with the need to ensure correct functionality, meet stringent safety and quality standards, and avoid repeated design iterations and errors [8], [10], [13]. In particular, existing approaches with a central controller to minimize effort are not suitable for time-critical systems with deterministic communication [17], [18].

Current approaches do not provide integrated methods for modeling and verifying reconfiguration of heterogeneous, mixed-criticality communication in distributed control applications. In particular, there is a lack of formal techniques that guarantee deterministic execution. In contrast, offline reconfiguration is less flexible but enables predefined, verified adaptations for known scenarios, ensuring predictable timing and deterministic behavior. For example, contract specifications are integrated as assume-guarantee (A/G) contracts into the application model to support a correct-by-construction design [1], [2]. Abstracting components with properties that specify behavioral aspects as contracts enables compositional verification and reuse of proof. This reduces the effort of formal methods while still offering a reliable verification.

This paper investigates the following question: *How can we enable predictable and reliable reconfiguration of real-time communication for heterogeneous network structures in distributed control systems?* Our contributions are:

(a) A design flow that supports modeling reconfigurable safety-critical systems (Sec. III).
(b) Modeling communication (Sec. IV): We explain how network communication models need to cover the complete communication stack from application layer to transport layer and physical layer. In particular, we establish a new guideline for integrating different communication protocols for heterogeneous network infrastructures.
(c) Early syntactic and semantic checks for formal contract specifications, as well as dynamic validation of generated time traces (Sec. V).

We demonstrate our approach for the case of battery production to illustrate both specification of different communication protocols using messages and channels and reconfiguration including a verification approach. Early syntax and semantic checks are based on a new extension for contract specification in the Eclipse 4diac Project (https://eclipse.dev/4diac/) as a

The contribution of L. Wünderlich and A. Rauh to this work has received funding from the European Union's Horizon Europe programme under grant agreement No. 101137954. L. Wünderlich and A. Rauh would like to thank the rest of the BATTwin consortium for supporting this research. B. Wiesmayr acknowledges the support by the LIT Project SOFIA (LIT2024-13-YOU-117) funded by the Linz Institute of Technology (LIT), the State of Upper Austria and the Federal Ministry of Education, Science and Research (BMBWF).

tool for developing distributed control software. Conclusions and an outlook on future work are given in Sec. VII.

II. Demonstration Scenario: Lithium-Ion Battery Cell Production

In this section, the battery production process [7], [12] is introduced as a demonstration example. Lithium-ion battery cell production involves multiple distributed processes that are coordinated across different machines and systems. The process steps consist of electrode manufacturing, cell assembly, and cell finishing. In general, electrode production and cell finishing are rather independent of the cell type.

The production process may require dynamic reconfiguration to adapt to changes in production volumes, product specifications, or maintenance schedules. Moreover, battery production demands precise control over environmental conditions (e.g., temperature, humidity) and process parameters to ensure product quality and safety. Lastly, many production steps require real-time monitoring and control, such as during electrolyte filling or cell finishing. Thus, network communication is required for seamless interaction and synchronization among different process steps and machines. Communication modeling can support flexible (re-)configuration of the system architecture to handle process adaptation efficiently. Real-time concerns can be addressed explicitly through communication modeling as the basis for selecting optimal communication protocols and schedules wrt. low-latency and high reliability. Verification and validation, particularly for network communication, ensures that data are exchanged accurately, reliably, and timely without delay between distributed components.

Electrode manufacturing is the first part of battery production and consists of six steps: (1) Mixing raw materials, (2) coating the foil with slurry, (3) drying the active material, (4) calendering the electrodes to a controlled thickness and porosity, and (5) slitting, before (6) vacuum drying. We select two steps as a more explicit application scenario, namely drying (3) and calendering (4), cf. Fig. 1. These are parts of a continuous process with close interaction and short cycle times that requires high precision. The execution schedule with an overall cycle time of 60 seconds for all functional parts as well as communication separates time-critical (red) from best-effort (yellow) traffic. Here, the different types of traffic are sent across separate networks. Messages are abbreviated as Message_Sender_Receiver, for example Message_SpeedControl_Drying (M_SD).

Through these production steps, the foil is transported by roller systems where a suitable foil pre-tensioning is important to avoid film tears. The line pressure influences the subsequent wetting properties and the energy density of the cell. In case the line pressure is set too high, squeezing may occur leading to stress cracks. The thickness of the foil is measured to confirm a high product quality. Next, the calendered mother rolls are (manually) transported to the slitting station.

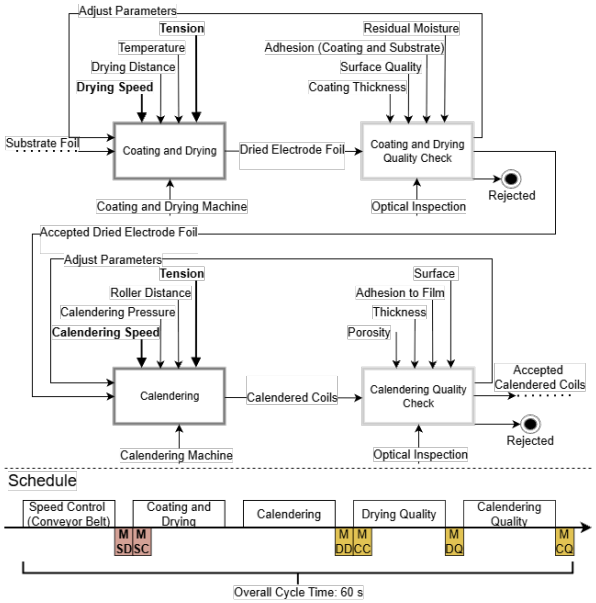

Fig. 1: Integrated DEFinition (IDEF0) model for the lithium-ion battery production steps coating/drying and calendering with quality checks (cf. [12]). Time-critical data is bold in the IDEF0 model; its transmission bold and red in the schedule.

III. Reconfiguration Strategy

Including reconfiguration plans within the design flow offers significant advantages for mixed-criticality real-time systems, particularly where predictability, safety, and certification are essential. By precomputing and verifying all valid system configurations before deployment, it is ensured that any switch between configurations, such as mode changes, preserves timing guarantees and system correctness. This eliminates the need for complex, computationally expensive decision-making at runtime, thereby reducing system overhead, avoiding unpredictability, and enabling lightweight runtime mechanisms. Finally, offline reconfiguration facilitates formal verification.

However, not all potentially needed reconfigurations are addressed in the initial design of complex industrial processes, e.g., a required fall-back solution in case of network failure when the coating/drying machine cannot communicate the current foil tension to the calendering machine anymore.

Fig. 2 presents a methodology to design reconfigurable manufacturing systems and follows a correct-by-construction design that includes iterative refinement steps with verification. The design does not only cover the application model and its mapping to a developed platform model but explicitly integrates communication and its mapping on a network. All information about the requirements, models, data or switching patterns are stored in a repository as a foundation for the reconfiguration. This can include switching between different operating modes, which have to be modeled explicitly. Only if different application models are stored in the repository, they can be accessed during system operation. Similarly, the hardware architecture is modeled and different configurations are stored. Most importantly, requirements and the different

models need to be verified continuously throughout the design process. Based on the repository, it becomes possible to react to reconfiguration requests from the operational phase. It is valid to switch between all modeled and verified system configurations online following a safe reconfiguration scheme that ensures that no unstable system state is reached when switching. However, in case there is no appropriate configuration available, it is necessary to model and verify additional applications, hardware architectures, and communication infrastructures with respect to the new requirements offline.

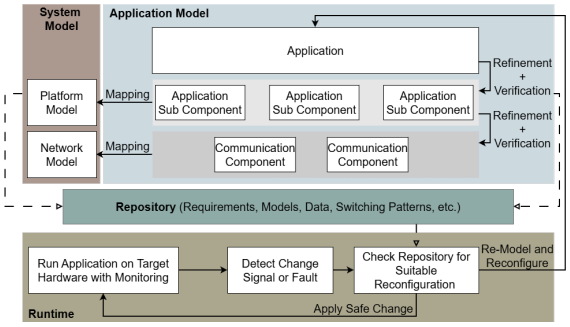

Fig. 2: Design flow: Initial development and reconfiguration.

This way, reconfigurations can be applied safely by reiterating through the design process and including additional models and their requirements. We present the modeling approach for network communication in Sec. IV. Since we consider a contract-based design, requirements are formalized as A/G contracts. This allows formally checking the consistency of a component-contract structure and reusing previously specified contracts. When modeling the complete distributed system including network communication explicitly, it becomes possible to verify the distributed control system completely (Sec. V).

IV. Model-Driven Engineering of Communication

When distributing an application onto several devices, it needs to be decided (depending on the application and network requirements) which communication protocols are used. The following general steps define modeling of communication:

1) Determine communication purpose: Define the immediate cause for communication within the system. Identify in detail what needs to be communicated and why.
2) Check for cross-layer dependencies: Here, it is necessary to verify that decisions made at one layer (e.g., physical, transport, or application) do not conflict with or constrain the requirements or capabilities of other layers.
3) Align with system model goals: Ensure that the chosen communication model supports the overarching goals of the entire system, such as performance, scalability, or safety. This step ensures that the communication aligns with the system's purpose and its higher-level objectives.

Categorizing where communication models have the most impact can be helpful to clarify whether a communication protocol applies to configuring network segments (system-level concerns), to specify transport layer concerns (communication patterns such as publish/subscribe), or to define logical messages (application-level concerns). The modeling standard IEC 61499 supports this by offering communication modeling elements at all three layers relying on the recently proposed concept of message function blocks (FBs) and channels [3]. Communication protocols that are applied at network and link layer typically influence network behavior and can be configured for network segments using channels. Transport layer protocols ensure reliable or efficient data delivery between endpoints and can be configured by communication FBs. Application layer protocols, which can be configured by using the message FB, directly influence logical message exchange within applications.

Table I provides an overview of different criteria, which represent the basis for deciding whether the selected communication protocol primarily concerns the infrastructure (network segment level), the transport layer, or the functionality of the application (message level). Communication protocols often imply cross-layer dependencies. These can be addressed by specifying the diverse parameters through the available modeling elements. It is possible to automatically derive a suitable transport layer mechanism based on the communication protocol specified at the application or system level (i.e., network layer and below). In particular, at the system level, we can derive the communicating devices, including their IP addresses, based on the network segment specification. Moreover, the channels and their properties, such as the schedule of a TSN network, can be identified from the system model. Additionally, at the application layer, we can retrieve the relevant information about the messages, which will be transmitted across their respective channels. In addition to the specific data that will be transmitted and their semantics, also protocol-specific information is relevant (such as the QoS level of a channel in case of an MQTT communication protocol).

For coating and drying as well as calendering in battery production, consistent drying has to be ensured to achieve the required electrode properties and a uniform calendering for proper thickness. Consequently, the quality of the final product depends on tight real-time control of process parameters such as temperature, pressure, and speed. A feedback control model could use communication protocols like OPC UA for communication between sensors, actuators, and controllers. Other goals are to minimize process delays while maintaining precision, or to provide real-time insights into process parameters and traceability for quality assurance. Achieving high throughput requires synchronization between process steps and minimal communication delays. In this case, TSN could be used to ensure deterministic communication between machines. This supports synchronized operation, e.g., drying machines notify the calendering units when the electrodes are ready. Lastly, for monitoring and traceability, real-time insights help operators identify bottlenecks, whereas traceability is crucial for quality checks. For drying, sensors can publish data such as temperature and duration and for calendering, thickness sensors can publish measurement data to a central control system via MQTT which allows selecting from three different QoS levels for data transmission. For this work,

TABLE I: Decision basis for modeling network protocols via modeling elements at different communication stack layers.

	Network Segment	Communication FBs	Logical Message FBs
Focus	Topology, routing, timing	Pattern, QoS	Semantic, payload, interaction
Layer	Physical, Data Link, Network	Transport	Application
Protocol	TSN, Ethernet, Wi-Fi	UDP, OPC UA Pub/Sub	MQTT, HTTP, CoAP
Scope	Components of the network	Comm. mechanism	Data flow

we chose the MQTT communication protocol next to the existing implementation of TSN [3]. Note, MQTT and TSN can be combined with OPC UA. MQTT is only active at the application layer, therefore, corresponding message types are defined for each of the QoS levels with a varying number of data ports. Then, MQTT messages can be mapped to a common Ethernet segment or to a selected channel in case of specific communication protocols such as TSN.

V. Supporting Correct-by-Construction Design by Checking Formal Specifications

Our work supports engineers in validating correctness of formal specifications, which are later used for the verification of distributed control applications. We specify timing behaviors as a set of A/G for each component using MTSL [1]. An assumption specifies the assumed periodic start of a code structure; if fulfilled, it is guaranteed to keep a specified time constraint from activation to completion. A top-level contract C can be refined into several sub-contracts C_n. Considering a contract C refined as C_a and C_b, the top-level contract is only fulfilled when the composition of C_a and C_b holds ($C \leq C_a \otimes C_b$). Thus, the contracts need to be compatible and the assumptions of one component must not be violated by another. For this, tool support is required to formally define timing properties of components, to validate the specification patterns, and to support a correct-by-construction design. Based on specified requirements, the application model is iteratively refined, mapped onto a hardware followed by continuous integration checks as shown in Fig. 2. In case of a distribution onto multiple devices, we consider communication as an integral refinement step of the application and hardware model that requires integration checks respectively.

An A/G contract for the coating and drying component of battery production can be expressed as follows relying on MTSL patterns: We assume that this component has an input event that occurs when its process starts and an output event that occurs once drying is finished. We start by specifying the components' behavior as a guarantee. Typically, in industrial-scale lithium-ion battery manufacturing drying times fall within the range of 20 to 60 seconds depending on the coating thickness [12]. For specifying the assumption of the contract, the behavior of the component's environment is taken into account. In our case, the second process step considered is calendering, which is much faster and could take between 10 and 20 seconds. A corresponding contract can be specified:

```
A: Drying.Start occurs every 60s with offset
   [25.5, 60.5]ms
G: Reaction(Drying.Start,Drying.Finish) within 20s
```

Here, the assumption states that the overall cycle time of the production pipeline under consideration is 60 seconds. This time was deliberately chosen for providing more flexibility to structuring the subcomponents. It is important to mention that drying is not the first step in this pipeline, but has to take into consideration that other process steps are executed first within each cycle. In this case, we consider the scan cycle for receiving the speed of the conveyor system as a given offset of [25.5, 60.5] milliseconds. Note that in contrast to the scan cycle of a system, the drying unit considers the physical process duration of coating and drying. Specifying the offset as an interval instead of a point-based value provides more flexibility in the execution, however, it comes with an increased complexity for the verification step. Then, a guarantee states that the drying component will emit the finish event exactly 20 seconds after it received the start event, indicating that the drying process is completed.

To perform checks based on the specified contract, we first have to verify that the contract rules adhere to the MTSL syntax and are semantically correct. The syntax definitions from the MTSL grammar [4] are implemented using Eclipse XText[1]. XText generates a parser that handles the syntax check, while the semantics are checked by custom written validators. They ensure that, e.g., the specified events exist as event pins within the system, that input and output events are only used in their respectively allowed places, or that any specified timing interval is properly defined (i.e., not empty).

Once the syntax and semantics of each rule have been checked, we perform further static checks between rules of a given system. These ensure that multiple rules do not contradict each other wrt. the expected timing behavior of events. To demonstrate such a check, we extend the example above with a conveyor belt that precedes drying and is linked to it via its event pins. We specify the contract for the conveyor belt component that transports the electrode foil and connect `Conv.Finish` to `Drying.Start` as follows:

```
A: Conv.Start occurs every 60s
G: Reaction(Conv.Start, Conv.Finish) within 5s
```

However, this means that the `Conv.Finish` event occurs every 60s with an offset of 5s, violating the contract specified for the `Drying.Start` event. Such errors can easily happen when modifying contracts during the development and forgetting to adapt certain properties, such as the reaction time in this case. Our static contract checker already catches such situations and reports the following error:

[ERROR] Drying assumes that the first event of Start occurs with an offset of [25.5,60.5]ms, but Conv only guarantees occurrence within 5s.

Real-time critical communication is explicitly modelled at application layer by using contracts in a similar matter. This

[1]https://eclipse.dev/Xtext/

is not needed for best-effort communication, such as MQTT messages. For example, for a time-critical message to transmit the current tension and speed, the start of the sending process M_CD.Start at the conveyor belt unit relates to receiving the data at the drying unit M_CD.Finish according to:

```
A: M_CD.Start occurs every 60s with offset
   [25.5, 60.5]ms
G: Reaction(M_CD.Start, M_CD.Finish) within 0.5ms
```

This static check is the first step towards dynamically checking timing correctness and formal verification that will follow. Such a contract checker can find violations based on a sequence of recorded events, either from a simulation, e.g. supported by the IDE, or based on a real system. The recorded event sequence is processed in order of the event occurrence timing and for each occurrence it is decided if it is valid or not. Each event is associated with a certain set of contract rules and for the dynamic check, each rule is associated with an internal state. This state keeps track of relevant information about event occurrences that were already processed and helps to decide if any new occurrence is valid. For example, the single event contract rule only allows one occurrence of the specified event. A Boolean flag stating if the event has already occurred is sufficient for checking validity in this situation. However, different contract rules such as repetitions or reactions require more sophisticated internal states and checking routines.

VI. Evaluation

We evaluate two different scenarios in the context of lithium-ion battery cell production: (1) a network fails and it is necessary to restructure its architecture, and (2) a mode switch occurs where the frequency of production is changed. The modeling elements, attaching timing contracts, and tool-supported verification are available within the Eclipse 4diac Project, enabling preliminary consistency checks, integrating model checking techniques, and dynamic validation that considers the complete application. An IEC 61499 standard compliant definition of the communication modeling elements is available as a white paper[2]. Thereby, our approach is open-source available and can by applied in industrial settings.

The system architecture consists of a device for controlling the conveyor system, the coating/drying process and the calendering step, which are connected via a TSN network and an MQTT network for communication to a supervisory quality control unit. This way, we consider the behavior of mixed-criticality real-time communication for low-level control according to the schedule shown in Fig. 1. Time-critical messages are mapped to the TSN network and all other messages are mapped to the MQTT network, transmitted with QoS level 1, because it provides reliable delivery and ensures that messages arrive at least once, which is usually acceptable for quality control signals where loss of a message is more critical than receiving it twice. A switched order of transmitting M_SD and M_SC results in a delayed consideration of the speed and tension of the material for coating/drying. This can lead to a massive loss of product quality. While this scenario represents only minimal timing difference, it demonstrates how changes in the time-critical schedule can lead to issues in the execution of a complex process.

In scenario (1), an MQTT network failed and communication is only possible via the TSN network. By adjusting the system model and mapping of the communication to the network, we can connect the supervisory control to the TSN network and use it to transmit MQTT messages. This reconfiguration does not influence the timing behavior because TSN is designed for transmitting concurrent non-real-time traffic. MQTT messages are mapped to previously empty buffer channels. However, the restructured system model and mapping must be stored in a common repository to ensure correct functionality through validation, see Fig. 2.

In scenario (2), an update of frequency occurs, e.g., because there is a decreased demand in battery cells. This requires a speed reduction of the conveyor, and consequently, a more relaxed control cycle. Changes are represented as an adjustment of the contract for the conveyor belt. It is assumed that an input event still occurs every 60 s, however, the guarantee specifies that whenever an input event occurs, the output event occurs within 30 to 65 ms. If the following subcontracts and the top-level contract do not take this offset into account correctly, they are inconsistent with the conveyor belt contract. The contract evaluation correctly detects this issue.

Even with contract evaluation support available when modeling an application, some issues might only become apparent at later design stages. Therefore, it is helpful to simulate possible execution time traces and validate whether contract specifications can be adhered to. Fig. 3a shows that event relations can be flagged in the execution time trace to highlight violated contracts. Here, the M_SC message FB takes longer for data transmission than expected. Therefore, the calendering component receives the data too late. In addition to the highlighted failed event as part of the time trace, error messages notify the developer about all contract violations. In contrast, Fig. 3b shows a valid time trace.

VII. Conclusions and Outlook

This work addressed the challenge of modeling and verifying reconfigurable mixed-criticality communication in industrial distributed control. Motivated by the increasing heterogeneity of communication networks in industrial cyber-physical systems, we proposed a methodology that integrates communication modeling directly into control software development. By extending IEC 61499 with explicit modeling for messages and channels, we enable the specification of timing, criticality, and communication protocols within a unified application model. A/G contracts are used to formalize communication requirements between components and support both compositional verification and early consistency checks. Our proposed workflow supports dynamic validation and offline verification, such as model checking, of communication correctness while allowing for predefined, reconfigurable system modes, ensuring deterministic behavior.

[2]https://eclipse.dev/4diac/doc/specs/index.html

(a) Failed execution time trace, because the message transmission of M_SC took too long.

(b) Valid execution time trace.

Fig. 3: Generated execution time traces to dynamically validate whether the timing specifications are kept.

Future work will extend handling dynamic reconfiguration scenarios online and further enhance tool automation for large-scale industrial systems. Additionally, different contract rules such as repetitions or reactions require more sophisticated internal states and checking routines as part of the validation process. Moreover, we aim to investigate how timing properties to be enforced on the deployed system can be systematically derived from functional and non-functional requirements.

REFERENCES

[1] F. Bruns, S. Mehlhop, B. Wiesmayr, and A. Zoitl. Enabling Automated Timing Verification: A Unified Approach for Industrial Distributed Control Systems. In *25th IEEE Intl. Conf. on Industrial Technology (ICIT)*, Bristol, UK, 2024.

[2] F. Bruns, J. Walter, and A. Rauh. Contract-Based Design for Robust Networked Control Systems in Industrial Automation. *ISA Transactions*, 167, 2025.

[3] F. Bruns, B. Wiesmayr, and A. Zoitl. Supporting Model-Based Network Specification for Time-Critical Distributed Control Systems in IEC 61499. In *19th IEEE Intl. Conf. on Automation Science and Engineering (CASE)*, Auckland, New Zealand, 2023.

[4] E. Böde, W. Damm, G. Ehmen, M. Fränzle, K. Grüttner, P. Ittershagen, B. Josko, B. Koopmann, F. Poppen, M. Siegel, and I. Stierand. MULTIC-Tooling, 2019.

[5] N. S. Bülbül, D. Ergenç, and M. Fischer. SDN-based Self-Configuration for Time-Sensitive IoT Networks. In *46th IEEE Conf. on Local Computer Networks (LCN)*, Caen, France, 2021.

[6] H. Chahed and A. J. Kassler. Software-Defined Time Sensitive Networks Configuration and Management. In *7th IEEE Conf.on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2021.

[7] O. Demir, M. Colledani, G. Horváth, D: Marzogui, and K. Giakoumi. A Novel Zero-Defect Manufacturing Approach for Defect Reduction in Battery Production. *Procedia CIRP*, 134, 2025.

[8] C. Eymüller, J. Hanke, A. Hoffmann, W. Reif, M. Kugelmann, and F. Grätz. RealCaPP: Real-Time Capable Plug & Produce Communication Platform with OPC UA Over TSN for Distributed Industrial Robot Control. In *17th IEEE Intl. Conf. on Automation Science and Engineering (CASE)*, Lyon, France, 2021.

[9] H. S. Fadhlillah, B. Wiesmayr, M. Oberlehner, R. Rabiser, and A. Zoitl. Towards Delta-Oriented Variability Modeling for IEC 61499. In *IEEE Intl. Conf. on Emerging Technologies and Factory Automation*, 2021.

[10] M. H. Farzaneh and A. Knoll. An Ontology-Based Plug-and-Play Approach for In-Vehicle Time-Sensitive Networking (TSN). In *7th IEEE Ann. Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Vancouver, BC, Canada, 2016.

[11] H.-M. Hanisch, M. Hirsch, D. Missal, S. Preuße, and C. Gerber. One Decade of IEC 61499 Modeling and Verification - Results and Open Issues. *IFAC Proceedings Volumes*, 42, 2009. 13th IFAC Symposium on Information Control Problems in Manufacturing.

[12] H. Heimes, A. Kamper, S. Wennemar, L. Plocher, G. Bcokey, S. Michaelis, and J. Schütrumpf. Production Process of a Lithium-Ion Battery Cell, 2023.

[13] S. Lucia, M. Kögel, and R. Findeisen. Contract-Based Predictive Control of Distributed Systems with Plug and Play Capabilities. *Inter. Federation of Automatic Control*, 48, 2015.

[14] J. Lunze and J. H. Richter. Entwurfsmethode für den verallgemeinerten virtuellen Aktor zur Rekonfiguration von Regelkreisen (Design Approach to the Generalised Virtual Actuator for Control Reconfiguration). *at - Automatisierungstechnik*, 54, 2006.

[15] D. Macherki, T. M. L. Diallo, A. Guizani, M. Barkallah, J.-Y. Choley, and M. Haddar. Effective Implementation of CPPS Self-reconfiguration Functionality: Research Review. In Lassaad Walha, Abdessalem Jarraya, Fathi Djemal, Mnaouar Chouchane, Nizar Aifaoui, Fakher Chaari, Moez Abdennadher, Abdelmajid Benamara, and Mohamed Haddar, editors, *9th Conf. on Design and Modeling of Mechanical Systems (CMSM)*, Hammamet, Tunisia, 2023. Springer International Publishing.

[16] L. Prenzel, S. Hofmann, and S. Steinhorst. Real-time Dynamic Reconfiguration for IEC 61499. In *5th IEEE Intl. Conf. on Industrial Cyber-Physical Systems (ICPS)*, 2022.

[17] B. Schneider, A. Zoitl, M. Wenger, and J. O. Blech. Evaluating Software-Defined Networking for Deterministic Communication in Distributed Industrial Automation Systems. In *22nd IEEE Intl. Conf. on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Cyprus, 2017.

[18] L. Silva, P. Pedreiras, P. Fonseca, and L. Almeida. On the Adequacy of SDN and TSN for Industry 4.0. In *22nd Inter. Symp. on Real-Time Distributed Computing (ISORC)*, Valencia, Spain, 2019.

[19] J. Wilch, B. Vogel-Heuser, F. Sax, S. Rüth, U. Oeckl, B. Wohlschläger, Y.-M. Hsieh, and F.-T. Cheng. Enhancing the Resilience of IEC 61131–3 Software With Online Reconfigurations for Fault Handling. *IEEE Trans. on Automation Science and Engineering*, 22, 2025.

[20] M. Wollschlaeger, T. Sauter, and J. Jasperneite. The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0. *IEEE Industrial Electronics Magazine*, 11, 2017.